ARCADA

Planering och utveckling av serverdelen i ett medlemsregister för specialföreningar

Christoffer Holmberg

Examensarbete
Informationsteknik
2015

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informations- och Medieteknik
Identifikationsnummer:	5009
Författare:	Christoffer Holmberg
Arbetets namn:	Planering och utveckling av serverdelen i ett medlemsregister för specialföreningar
Handledare (Arcada):	Göran Pulkkis
Uppdragsgivare:	Tekniska Läroverkets Kamratförbund r.f.

Sammandrag:
Medlemshantering för specialföreningar är något som ofta sker i Excel-tabeller eller i värsta fall med papper och penna. Detta examensarbete ger ett alternativ till de gamla metoderna genom en specifikt för specialföreningar anpassad databas och ett enkelt webbgränssnitt. Examensarbetet består av planering, programmering och installation av serverdelen och en dokumentationsdel där serverns användning beskrivs samt några exempel på hur man kan kommunicera med serverns gränssnitt.

Nyckelord:	Medlemsregister, Databas, Python, Django, REST
Sidantal:	43
Språk:	Svenska
Datum för godkännande:	3.6.2015

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	5009
Author:	Christoffer Holmberg
Title:	Planning and development of a server for a member directory for student associations
Supervisor (Arcada):	Göran Pulkkis
Commissioned by:	Tekniska Läroverkets Kamratförbund r.f.

Abstract:
Member management for student associations is usually done with Excel spreadsheets or even with pen and paper. The purpose of this work is to act as an alternative to old methods and supply a database customized for the needs of student associations and a simple web interface. The work consists of two parts. Design, programming and installation of the server and the documentation on how to use the server and examples on how to communicate with the server interface are included.

Keywords:	Member Directory, Database, Python, Django, REST
Number of pages:	43
Language:	Swedish
Date of acceptance:	3.6.2015

INNEHÅLL

FIGURER

FÖRKORTNINGAR OCH BEGREPP

API = Application Programming Interface

IDE = Integrated Development Environment

HTTP = Hypertext Transfer Protocol

JSON = JavaScript Object Notation

REST = Representational State Transfer

SSL = Secure Sockets Layer

URL = Uniform Resource Locator

WSGI = Web Server Gateway Interface

Frontend = Användargränssnitt

Backend = Śervernivå

1 INLEDNING

TLK har redan en tid behövt ett fungerande elektroniskt medlemsregister. Tillsammans med en annan studerande har jag planerat ett databasbotten för detta och nu skulle det vara dags för en komplett backend/frontend lösning för att hantera all medlemsdata på ett effektivt sätt.Detta projekt skall ge nytta till TLK samt till andra specialföreningar om de anser att de behöver det.

1.1 Arbetssätt

Jag kommer att hämta en stor del av svaren till mina programmeringsproblem med Google-sökningar samt från dokumentationen för de ramverk och programmeringsspråk jag använder.

1.2 Beställaren

> TLK grundades i januari 1917, för att främja kamratskapet bland studeranden vid Tekniska Läroverket i Helsingfors. Sedan dess har det skett en del strukturella förändringar, men verksamhetsprincipen är ännu den samma.
>
> Alla ingenjörsstuderanden vid Arcada, Nylands svenska yrkeshögskola, har rätt att ansluta sig till TLK. TLK har ca. 120 studerande medlemmar och ett större antal äldre utdimitterade ständiga medlemmar. En TLK-medlem kan också bli medlem i Arcada studerandekår – ASK, som TLK samt de övriga ämnesföreningarna i Arcada samarbetar med.

(Tekniska Läroverkets Kamratförbund r.f. TLK)

1.3 Syfte och mål

Målet är att producera en fungerande backend och frontend för beställaren. Det skall också vara möjligt att utveckla andra frontends med hjälp av applikationens programmeringsgränssnitt (API) i detta projekt.
Frågor som skall besvaras:
Vilket ramverk och vilket programmeringsspråk lämpar sig bäst för backenden samt frontenden?
Hur implementerar jag bäst min beställares krav på projektet?
Hur bra kan jag uföra ett projekt i denna skala.

1.4 Avgränsning

I detta arbete behandlas utvecklingen av medlemsregistrets serverdel.

2 PLANERING

Här tas upp hur planeringen gick till och på vilket sätt jag fått fram vilka egenskaper behövs.

2.1 Egen uppfattning

Efersom jag själv suttit i TLK:s styrelse har jag rätt så fria händer när det kommer till planeringen. Min plan för medlemsregistret är att bygga en server-klient-lösning. Servern sköter databasen och användarhanteringen och erbjuder en REST-tjänst som ger en frihet att utveckla olika klienter enligt behov. Serverns REST API är bläddringsbar och går att använda som sådan med en webbläsare eller med en för ändamålet utvecklad klient. Idén

bakom detta är att om klienten inte längre uppfyller beställarens behov kan man lätt göra en annan klient som är bättre lämpad.

2.2 Intervju med uppdragsgivaren

Efter diskussion med TLKs styrelse har vi kommit fram till att TLK behöver en medlemsdatabas där nya medlemmar lätt kan tilläggas och där snabba sökningar på diverse medlemsdata kan göras.

3 VERKTYGEN

Här beskrivs de programmeringsspråk, verktyg och programvara som använts.

3.1 Programmeringsspråk

Före detta projekt var mina Python-kunskaper på nybörjarnivå. För att få en djupare uppfattnning om språket utförde jag Python-kursen på codecademy.com (Codecademy 2015). Med denna kunskap som botten och med hjälp av två ramverk har jag lyckats utveckla en fungerande backend.

3.2 Ramverk

Användning av ramverk kan göra utveckling av program snabbare och ger programmeraren en möjlighet att koncentrera sig på den egentliga funktionaliteten i ett program istället för att använda tid på hur data skall representeras av programmet. (DocForge 2015)

3.2.1 Django

Django är ett webb-ramverk som fokuserar på snabb uveckling och pragmatisk design. Det tar hand om interaktionen till databaser och administration av användare. Användargränssnitt produceras med hjäp av mallar. Django är ett mycket skalbart, mångsidigt och säkert ramverk vilket gör det till ett bra val som grund för detta projekt. (Django Software Foundation 2015b)

En av Djangos fördelar är möjligheten att använda insticksmoduler (eng. plugin) som ytterligare utökar Djangos funktionalitet. Detta möjliggör snabb utveckling av ny funktionalitet vid behov. Ett sådant insticksprogram är Django Rest Framework.

3.2.2 Django Rest Framework (DRF)

DRF är ett verktyg för enkel och snabb utveckling av webb-API:n. DRF har färdiga metoder för att representera data med Django-databasmodeller och ger breda möjligheter för egen anpassning. Möjligheten till en bläddringsbar API är en stor fördel. DRF är en av de populäraste insticksmodulerna för Django och är ännu under aktiv utveckling. (Christie 2015)

3.3 Verktyg

För att kunna utveckla en bra produkt krävs det bra verktyg. I detta projekt har jag valt att använda verktyg som allmänt anses vara beprövade på sitt område.

3.3.1 Apache och mod_wsgi

Apache är en av de mest använda webbservarna (W3Techs 2015). Apache stöder ett brett urval av plattformer och moduler. För att kunna köra Django-projekt i Apache behövs modulen mod_wsgi. Web Server Gateway Interface (WSGI) är ett universellt gränssnitt mellan webbservrar och projekt programmerade i Python (Eby 2010).

WSGI agerar som en mellanhand för webservern och Pythonprogram och hanterar de anrop som webbservern förmedlar till Pythonprogrammet. WSGI förmedlar svaren från Pythonprogrammet tillbaka till webbservern.

3.3.2 Databas

För att lagra data i ett medlemsregister behövs en databas. Django stöder flere databasmotorer såsom t.ex MySQL, PostgeSQL och SQLite. Val av databasmotor bör göras på basen av data man vill lagra samt hur man använder databasbotorn och lagrad data. (Django Software Foundation 2015a)

3.4 JavaScript Object Notation (JSON)

JSON är ett enkelt dataformat för överföring av dataobjekt mellan olika gränssnitt. Ett JSON-objekt byggs upp av nyckel-värde-par med välkända datatyper. Ett exempel visas i Figur 1. (Ecma International 2013)

3.4.1 Integrerad utvecklingsomgivning (IDE)

En IDE finns till för att göra utvecklingsarbetet enklare. För utveckling i Python finns PyCharm. PyCharm har även stöd för ramverket Django och på så vis underlättas utveck-

```
{
"key": "value",
"floatkey": 1.23,
"boolean": true,
"array": [
          "key": "value",
          "otherkey": "othervalue"
         ],
}
```

Figur 1. JSON-exempel

lingsarbetet. (JetBrains 2015)

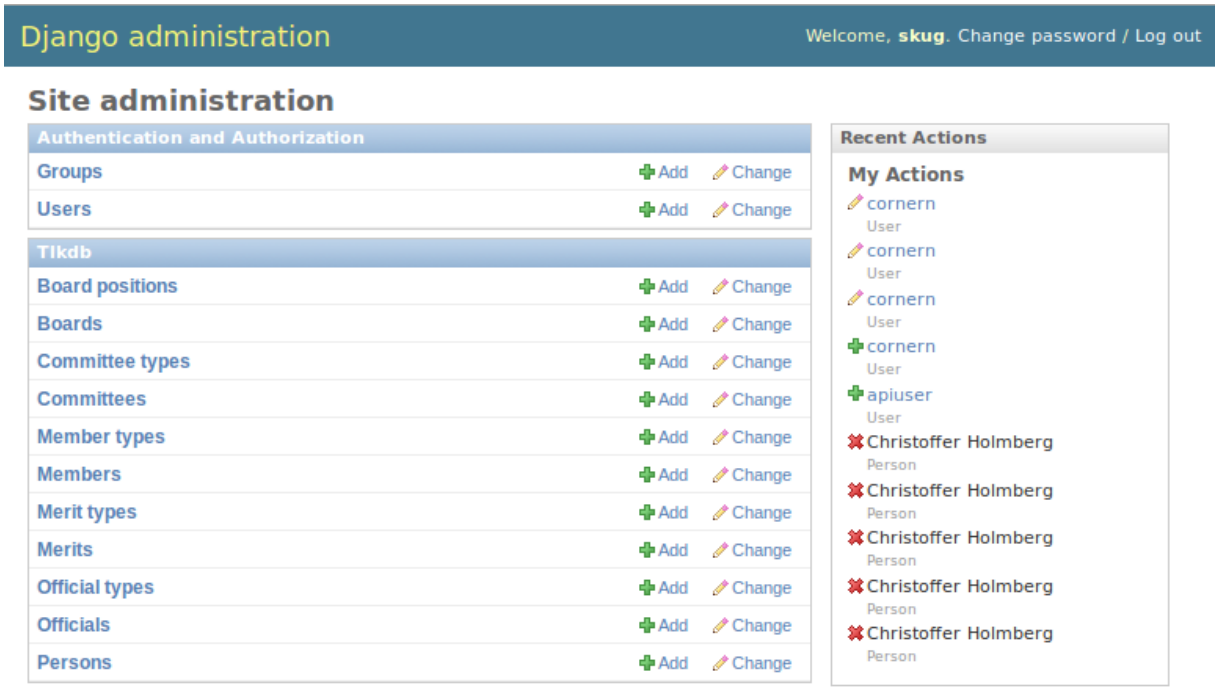
3.4.2 Versionshantering

För att få en överblick av ändringar man gjort under utvecklingsarbetet kan man använda ett versionshanteringsprogram. Jag har valt att avända Git i samband med Github. Github är en webbtjänst för versionshantering och buggsökning i projekt. Github är främst inriktad på kollaborativ utveckling men kan också användas om man jobbar ensam. (Git-scm (2015); GitHub Inc. (2015))

4 SERVER

4.1 Django admin

En av fördelarna med att använda Django är modulen admin som ingår i Django. Admin-gränssnittet gör det möjligt att direkt göra ändringar i databastabellerna och definiera användarkonton för en Django-app. Admin-gränssnittet är också behändigt när man definierar de grundtabeller som senare används för att koppla medlemsdata till personer. Exmpel på gränssnittet visas i Figur 2.

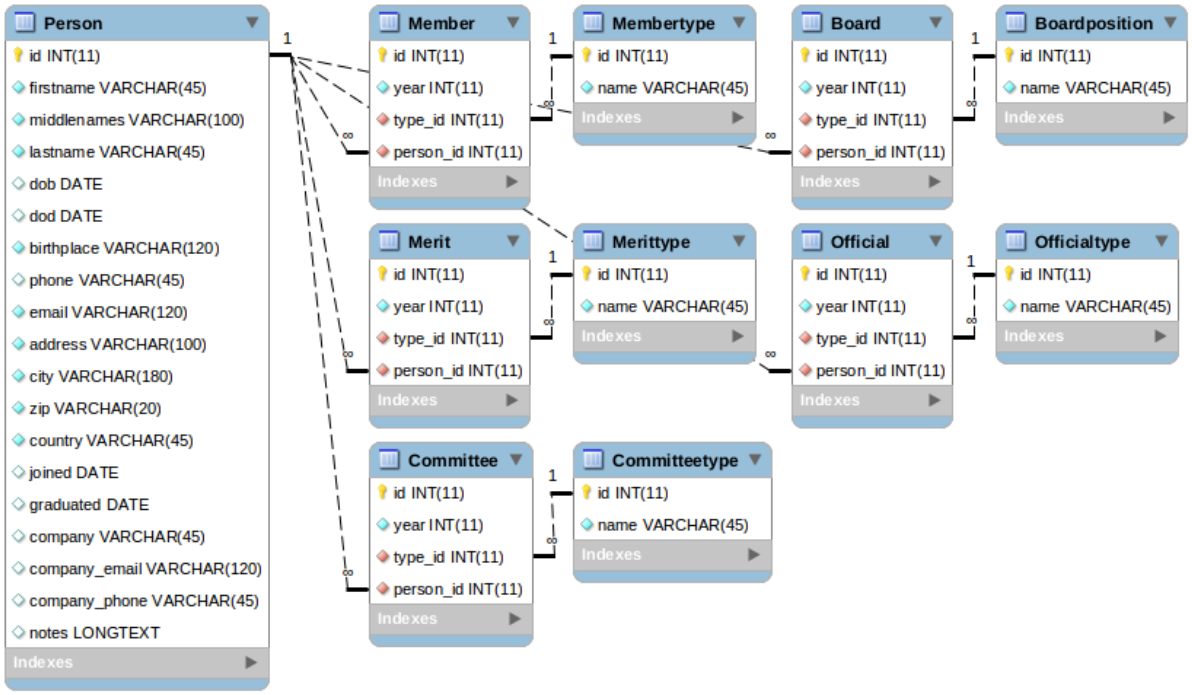


Figur 2. Exempel på admin-gränssnittet i Django.

4.2 Databasens struktur

Databasen byggs upp av en Person-tabell som refererar till flera undertabeller för mera information om medlemskap, styrelseposter, förtjänsttecken, funktionärsposter och utskott.

Varje undertabell har en referens till en tabell som innehåller de olika typerna för medlemskap etc. Databasdiagrammet är representerat i Figur 3.



Figur 3. Databasdiagram.

4.2.1 ”Model”-klasser i Django

Databastabellerna i Django definieras av ”Model”-klasser. Klassernas ordningsföljd har betydelse så långt att om en klass refererar till en annan bör den definieras efter den klass som refereras till. Exempel på en ”Model”-klass visas i Figur 4.

4.3 Användning av DRF

DRF består av ”Serializer”- och ”View”-klasser som tillsammans med ”Model”-klasserna i Django bygger upp REST-tjänsten. En ”View”-klass använder en ”Serializer”-klass som i sin tur hämtar data ur databasen enligt en ”Model”-klass. Exempel på en ”Serializer”-

```
class Member(models.Model):
    year = models.IntegerField()
    type = models.ForeignKey(MemberType, related_name='members')
    person = models.ForeignKey(Person, related_name='members')

    def __str__(self):
        return '%s %s %s' % (self.year, self.type, self.person)
```

Figur 4. Django "Model"-klass som definierar Member-tabellen.

klass visas i Figur 5.

```
class MemberAddSerializer(serializers.ModelSerializer):
    type = serializers.SlugRelatedField(read_only=False, slug_field='pk',
                                        queryset=MemberType.objects.all())

    class Meta:
        model = Member
        fields = ('pk', 'person', 'year', 'type')
```

Figur 5. Serialiseringsklass för DRF

4.3.1 Gränssnittet till klienten

Kommunikation med API:n sker via API-ändpunkter som definieras i en "View"-klass. En API-ändpunkt är en URL-adress som byggs upp av en bas-URL tex. `https://member.tlk.fi/` och den specifika änpunkten:

- `persons/` är en förteckning på alla personer i databasen med all underinformation om medlemskap. Denna ändpunkt är ämnad för hämtning av data och inga ändringar bör göras via den.

- `persons/add/` tar emot en JSON-sträng med ett personobjekt för att läggas in i databasen. En unik person-id genereras för varje ny person.

- `persons/{id}/` tar emot ett personobjekt med ändringar eller ett "HTTP DELETE"-

kommando för att radera personen med all tillhörande data ur databasen. `{id}` representerar en unik id för en person.

- `members/` är en förteckning på alla medlemsår för varje medlem. Identifiering av vilken person som varit medlem sker via en unik person-id och medlemstypen via en unik ”membertype”-id.

- `boards/` är en förteckning på alla styrelseposter per år och person. Identifiering av person och styrelsepost sker via unika värden på person-id och ”boardtype”-id.

- `committees/` är en förteckning på alla utskottsmedlemmar per år. Identifiering av person och utskottstyp sker via unika värden på person-id och ”officialtype”-id.

- `merits/` är en förteckning på alla utgivna förtjänsttecken per år. Identifiering av mottagare och förtjänsteckentyper sker via unika värden på person-id och ”merittype”-id.

- `officials/` är en förteckning på alla funktionärer per år. Identifiering av funktionärer och funktionärstyper sker via unika värden på person-id och ”officialtype”-id.

  Som ändpunkter fungerar även `add/` och `{id}/`. Funktionaliteten är samma som för `persons`.

- `membertype/` visar de tillgängliga medlemstyperna.

- `boardtype/` visar de tillgängliga styrelseposterna.

- `committeetype/` visar de tillgängliga utskotten.

- `merittype/` visar de tillgängliga förtjänstteckentyperna.

- `officialtype/` visar de tillgängliga funktionärsposterna.

För de fem sistnämnda ändpunkterna behövs inte ett tillägg `add/` för att lägga till nya typer av medlemmar, styrelseposter, utskott, förtjänsttecken och funktionärer. För att radera eller ändra en typ används tillägget `{id}`.

4.4 Klientexempel

För alla ändpunkter krävs det att man i ett HTTP-anrop skickar ett autentiseringshuvud som innehåller användarnamn och lösenord kodat i base64. Som klient kan användas en vanlig webbläsare för den bläddringsbara API:n eller en för ändamålet programmerad klient.

Exempel på HTTP-GET-anrop med information om person med id 15 som svar visas i figurerna 6 och 7. Svaret består i detta fall av ett JSON-objekt på personen med id 15.

Exempel på ett HTTP-PUT-anrop med svar för att ändra informationen om en person visas i figurerna 8 och 9. Till servern skickas ett JSON-objekt med den nya informationen och som svar fås ett JSON-objekt med samma data.

Exempel på ett HTTP-DELETE-anrop med svar för radering av informationen om en person med en given id visas i figurerna 10 och 11. Svaret består av HTTP-statuskoden ”204 NO CONTENT”.

4.4.1 Sökning och filtrering

För att göra fritextsökning på en ändpunkt kan man förlänga ändpunkten med `?search={sökord}`. Svaret är ett eller flera JSON-objekt där sökordet träffar. Denna fritextsökning stöds av ändpunkterna `persons/`, `members/`, `boards/`, `committees/`, `officials/` och `merits/`.

Filtrering ger en exaktare metod att finna det man behöver under en ändpunkt. För att filtrera `persons/` på alla personer som har förnamnet Kalle och bor i Esbo blir adressen `?firstname=Kalle&city=Esbo`

4.5 Installation av medlemsregistrets serverdel

Servern tas ibruk genom att klona Github-projektet TLKmAPI (Holmberg 2015). Efter kloning bör man sätta upp servermiljön att köra Python-projekt i Apache. TLKmAPI är beroende av följande programpaket i Linux:
`apache2, python3, libapache2-mod-wsgi, python-virtualenv, git.`
I Debian GNU/Linux kan dessa programpaket installeras med kommandot
`apt-get install apache2 python3 libapache2-mod-wsgi python-virtualenv git.`

4.5.1 Python virtualenv

Efter installation av programpaketen bör man skapa en Python-virtualmiljö i samma mapp som projektet TLKmAPI. Detta sker enklast med kommandot
`virtualenv -p /usr/bin/python3 venv.`
Sedan skall virtualmiljön aktiveras med kommandot
`source ./venv/bin/activate.`

4.5.2 Python paket

I projektet TLKmAPI finns en fil `requirements.txt` som innehåller en lista på de programpaket som bör installeras i virtualmiljön. Installation av programpaketen sker med kommandot
`pip install -r requirements.txt.`

```
GET /persons/15/?format=json HTTP/1.1
Host: member.tlk.fi
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:37.0) Gecko/20100101 Firefox/37.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: null
Accept-Encoding: gzip, deflate
DNT: 1
Authorization: Basic c2t1Zzptb2xuZXQ=
Cookie: csrftoken=BRwJCDyXqciRGM7IubXDvKuiNO1Va0ly; sessionid=d9gqcbqfl4w888dsydd98xbzy5595eyk
Connection: keep-alive
```

Figur 6. GET-anrop

```
HTTP/1.1 200 OK
Date: Tue, 19 May 2015 06:33:42 GMT
Server: Apache/2.2.22
Vary: Accept,Cookie
X-Frame-Options: SAMEORIGIN
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/json

{
"pk":15,
"firstname":"Another",
"middlenames":"Random",
"lastname":"Person",
"dob":"1988-01-24",
"dod":null,
"birthplace":"Someplace",
"phone":"",
"email":"somerguy@arcada.fi",
"address":"Someroad 12",
"city":"Somecity",
"zip":"12345",
"country":"Finland",
"joined":"2009-09-01",
"graduated":null,
"company":"Initech Oy",
"company_email":"herp@initech.fi",
"company_phone":"+358123456",
"notes":"Great Guy!",
"members":[],"boards":[],"officials":[],"merits":[],"committees":[]
}
```

Figur 7. GET-svar

Pip är Pythons officiella pakethanterare.

```
PUT /persons/add/15/?format=json HTTP/1.1
Host: member.tlk.fi
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:37.0) Gecko/20100101 Firefox/37.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: null
Accept-Encoding: gzip, deflate
DNT: 1
Authorization: Basic c2t1Zzptb2xuZXQ=
Content-Type: application/json; charset=UTF-8
Content-Length: 398
Cookie: csrftoken=BRwJCDyXqciRGM7IubXDvKuiNO1Va0ly; sessionid=d9gqcbqfl4w888dsydd98xbzy5595eyk
Connection: keep-alive

{
"pk":15,
"firstname":"Edited",
"middlenames":"Name",
"lastname":"Person",
"dob":"1988-01-24",
"dod":null,
"birthplace":"Someplace",
"phone":"",
"email":"somerguy@arcada.fi",
"address":"Someroad 12",
"city":"Somecity",
"zip":"12345",
"country":"Finland",
"joined":"2009-09-01",
"graduated":null,
"company":"Initech Oy",
"company_email":"herp@initech.fi",
"company_phone":"+358123456",
"notes":"Great Guy! More Info"
}
```

Figur 8. PUT-anrop

```
HTTP/1.1 200 OK
Date: Tue, 19 May 2015 06:49:48 GMT
Server: Apache/2.2.22
Vary: Accept,Cookie
X-Frame-Options: SAMEORIGIN
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/json

{
"pk":15,
"firstname":"Edited",
"middlenames":"Name",
"lastname":"Person",
"dob":"1988-01-24",
"dod":null,
"birthplace":"Someplace",
"phone":"",
"email":"somerguy@arcada.fi",
"address":"Someroad 12",
"city":"Somecity",
"zip":"12345",
"country":"Finland",
"joined":"2009-09-01",
"graduated":null,
"company":"Initech Oy",
"company_email":"herp@initech.fi",
"company_phone":"+358123456",
"notes":"Great Guy! More Info"
}
```

Figur 9. PUT-svar

```
DELETE /persons/add/15/?format=json HTTP/1.1
Host: member.tlk.fi
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:37.0) Gecko/20100101 Firefox/37.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: null
Accept-Encoding: gzip, deflate
DNT: 1
Authorization: Basic c2t1Zzptb2xuZXQ=
Content-Length: 0
Content-Type: text/plain; charset=UTF-8
Cookie: csrftoken=BRwJCDyXqciRGM7IubXDvKuiNO1Va0ly; sessionid=d9gqcbqfl4w888dsydd98xbzy5595eyk
Connection: keep-alive
```

Figur 10. DELETE-anrop

```
HTTP/1.1 204 NO CONTENT
Date: Tue, 19 May 2015 06:57:55 GMT
Server: Apache/2.2.22
Vary: Accept,Cookie
X-Frame-Options: SAMEORIGIN
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/x-python
```

Figur 11. DELETE-svar

4.5.3 Apache

För att Apache skall kunna vara värd för projektet TLKmAPI behövs en konfigureringsfil i `/etc/apache2/sites-available/`. Ett exempel visas i Figur 12.

Exempelfilen sparas som `/etc/apache2/sites-available/TLKmAPI.conf` och kan aktiveras med kommandot `a2ensite` TLKmAPI. För att Apache skall kunna använda mod_wsgi bör modulen vara aktiverad med kommandot `a2enmod wsgi`. Till sist bör man omstarta Apache med kommandot `apache2ctl restart`.

4.5.4 Säkerhet och SSL

Denna examenarbetsrapport beskriver endast hur man installerar projektet TLKmAPI utan säker dataöverföring. Vid fall då denna serverapplikation skall användas över Internet bör man se till att kommunikation med servern sker över en förbindelse säkrad med SSL.

```
<VirtualHost *:80>
ServerName exempel.server.tld
DocumentRoot /srv/TLKmAPI

Alias /favicon.ico /srv/TLKmAPI/TLKmAPI/static/favicon.ico
Alias /static/rest_framework/ /srv/TLKmAPI/virtualenv/lib/python3.2/site-packages/rest_framework/static/rest_framework/
Alias /static/admin/ /srv/TLKmAPI/virtualenv/lib/python3.2/site-packages/django/contrib/admin/static/admin/

<Directory /srv/TLKmAPI/virtualenv/lib/python3.2/site-packages/rest_framework/static/>
    Order Deny,Allow
    Allow from all
</Directory>

<Directory /srv/TLKmAPI/>
    Order Deny,Allow
    Allow from all
</Directory>

WSGIDaemonProcess TLKmAPI user=www-data group=www-data threads=5 python-path=/srv/TLKmAPI:/srv/TLKmAPI/virtualenv/lib/python3.2/site-packages
WSGIScriptAlias / /srv/TLKmAPI/TLKmAPI/wsgi.py
WSGIProcessGroup TLKmAPI
WSGIApplicationGroup %{GLOBAL}
WSGIPassAuthorization On

<Directory /srv/TLKmAPI/TLKmAPI>
    Order Deny,Allow
    Allow from all
</Directory>

ErrorLog ${APACHE_LOG_DIR}/TLKmAPI-error.log
LogLevel warn
CustomLog ${APACHE_LOG_DIR}/TLKmAPI-access.log combined
</VirtualHost>
```

Figur 12. Exempel på ”TLKmAPI.conf”

5 DISKUSSION OCH SLUTSATSER

Planen med projeketet var att skapa ett medlemsregister för specialföreningar och samtidigt se hurdana utmaningar uppstår när man startar ett projekt i ett främmande programmeringsspråk.

Min tanke i början av projektet var att programmering i ett nytt språk skulle vara både intressant och utmanande. Jag har en längre tid varit mycket nyfiken på Python men inte haft en tillräckligt bra orsak att lära mig. Nu med facit i handen kan jag konstatera att jag är nöjd med såväl mitt arbetssätt och tidsplanering som med själva slutresultatet.

Programmering i ett nytt programmeringsspråk för den här storlekens projekt gick bättre än förväntat. Om jag måste peka ut ett område som skapade mig problem så skulle det vara autentiseringen över domängränser. Speciellt nöjd är jag med de nya kunskaperna jag fått i Pyhton och Django.

Beträffande skrivandet av själva examensarbetsrapporten måste jag erkänna att flytande text inte är min starkaste sida men jag känner absolut att jag även på den fronten lärt mig mycket.

5.1 Varför TLKmAPI?

Om man googlar efter existerande medlemshanteringsmjukvara hittar man en hel del program som utför nästan samma funktionalitet som TLKmAPI. TLKmAPI är ett projekt menat specifikt för specialföreningar och deras krav på medlemshantering. TLKmAPI distribueras som öppen källkod under GPLv3 licens och ger specialföreningar ytterligare möjlighet att anpassa TLKmAPI för deras egna behov. Som jämförelse till TLKmAPI har jag hittat MASS (SoftSys 2015) och Wild Apricot (Wild Apricot Inc. 2015).

MASS är medlemshanteringsmjukvara med ett brett urval av egenskaper samt olika versioner för en lokal installation i Windows miljö och en webbversion. MASS går att anpassa till kundens behov. Nackdelen med MASS med tanke på små specialföreningar är kostnaden av själva mjukvaran.

Wild Apricot är en webbtjänst som erbjuder gratis medlemshantering för upp till 50 medlemmar. Wild Apricot är anpassningsbar enligt behov och medlemsregistret är sparat i en molntjänst. Medlemsregistrets placering i en molntjänst är inte en bra idé med tanke på datasäkerhet och personskydd.

TLKmAPI sparar all information i en lokalt och är gratis att använda. Detta gör TLKmAPI till ett bättre alternativ för specialföreningar.

KÄLLOR

Christie, Tom. 2015, *Django REST Framework*. Tillgänglig: http://www.django-rest-framework.org, Hämtad: 9.4.2015.

Codecademy. 2015, *Learn to program in Python, a powerful language used by sites like YouTube and Dropbox.* Tillgänglig: http://www.codecademy.com/tracks/python, Hämtad: 18.4.2015.

DocForge. 2015, *Framework*. Tillgänglig: http://docforge.com/wiki/Framework, Hämtad: 18.4.2015.

Eby, P.J. 2010, *Python Web Server Gateway Interface v1.0.1*. Tillgänglig: https://www.python.org/dev/peps/pep-3333/, Hämtad: 18.4.2015.

Django Software Foundation. 2015a, *Migrations*. Tillgänglig: https://docs.djangoproject.com/en/1.8/topics/migrations/, Hämtad: 18.4.2015.

Django Software Foundation. 2015b, *The Web framework for perfectionists with deadlines*. Tillgänglig: https://www.djangoproject.com, Hämtad: 9.4.2015.

Git-scm. 2015, *Git*. Tillgänglig: http://git-scm.com, Hämtad: 9.4.2015.

Holmberg, Christoffer. 2015, *TLKmAPI*. Tillgänglig: https://https://github.com/mrskug/TLKmAPI, Hämtad: 4.5.2015.

GitHub Inc. 2015, *GitHub*. Tillgänglig: http://github.com, Hämtad: 9.4.2015.

Ecma International. 2013, *ECMA-404 The JSON Data Interchange Format*. Tillgänglig: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, Hämtad: 28.4.2015.

JetBrains. 2015, *Jet Brains PyCharm 4.1*. Tillgänglig: https://www.jetbrains.com/pycharm, Hämtad: 9.4.2015.

SoftSys. 2015, *MASS*. Tillgänglig: http://www.membershipadmin.com.au, Hämtad: 22.5.2015.

Tekniska Läroverkets Kamratförbund r.f. (TLK). 2015, *Presentation / Historia*. Tillgänglig: http://www.tlk.fi/?page_id=66, Hämtad: 9.4.2015.

W3Techs. 2015, *Usage Statistics and Market Share of Web Servers for Websites, April 2015*. Tillgänglig: http://w3techs.com/technologies/overview/web_server/all, Hämtad: 18.4.2015.

Wild Apricot Inc. 2015, *Wild Apricot*. Tillgänglig: http://www.wildapricot.com, Hämtad: 22.5.2015.

6 BILAGA

```
"""
TLKmAPI/TLKdb/admin.py
"""
from django.contrib import admin

from TLKdb.models import *

# Registrera URLs for admin sidan
admin.site.register(Person)
admin.site.register(Member)
admin.site.register(MemberType)
admin.site.register(Merit)
admin.site.register(MeritType)
admin.site.register(Committee)
admin.site.register(CommitteeType)
admin.site.register(Official)
admin.site.register(OfficialType)
admin.site.register(Board)
admin.site.register(BoardPosition)
```

```
"""
TLKmAPI/TLKdb/models.py
"""
# coding=utf-8
from django.db import models


# Medlemstyper
class MemberType(models.Model):
    name = models.CharField(max_length=45)

    def __str__(self):
        return self.name


# Styrelseposter
class BoardPosition(models.Model):
    name = models.CharField(max_length=45)
```

```
    def __str__(self):
        return self.name


# Utskott
class CommitteeType(models.Model):
    name = models.CharField(max_length=45)

    def __str__(self):
        return self.name


# Funktionarer
class OfficialType(models.Model):
    name = models.CharField(max_length=45)

    def __str__(self):
        return self.name


# Fortjansttecken
#(Guld, Silver, SGRS, Arets Werkeit, Storwerket, Hedersmedlem)
class MeritType(models.Model):
    name = models.CharField(max_length=45)

    def __str__(self):
        return self.name

# Personer
class Person(models.Model):
    firstname = models.CharField(max_length=45)
    middlenames = models.CharField(max_length=100, blank=True)
    lastname = models.CharField(max_length=45)
    dob = models.DateField(null=True, blank=True)
    dod = models.DateField(null=True, blank=True)
    birthplace = models.CharField(max_length=120, blank=True)
    phone = models.CharField(null=True, max_length=45, blank=True)
    email = models.CharField(max_length=120)
    address = models.CharField(max_length=100, blank=True)
    city = models.CharField(max_length=180, blank=True)
    zip = models.CharField(max_length=20, blank=True)
    country = models.CharField(max_length=45, blank=True)
    joined = models.DateField(null=True, blank=True)
    graduated = models.DateField(null=True, blank=True)
    company = models.CharField(null=True, max_length=45, blank=True)
```

```
    company_email = models.CharField(null=True, max_length=120, blank=True)
    company_phone = models.CharField(null=True, max_length=45, blank=True)
    notes = models.TextField(null=True, max_length=1000, blank=True)

    def __str__(self):
        return '%s %s' % (self.firstname, self.lastname)

# Medlemmar
class Member(models.Model):
    year = models.IntegerField()
    type = models.ForeignKey(MemberType, related_name='members')
    person = models.ForeignKey(Person, related_name='members')

    def __str__(self):
        return '%s %s %s' % (self.year, self.type, self.person)

# Styrelseposter
class Board(models.Model):
    year = models.IntegerField()
    type = models.ForeignKey(BoardPosition, related_name='boards')
    person = models.ForeignKey(Person, related_name='boards')

    def __str__(self):
        return '%s %s %s' % (self.year, self.type, self.person)

# Funktionarer
class Official(models.Model):
    year = models.IntegerField()
    type = models.ForeignKey(OfficialType, related_name='officials')
    person = models.ForeignKey(Person, related_name='officials')

    def __str__(self):
        return '%s %s %s' % (self.year, self.type, self.person)

# Fortjansttecken
class Merit(models.Model):
    year = models.IntegerField()
    type = models.ForeignKey(MeritType, related_name='merits')
    person = models.ForeignKey(Person, related_name='merits')

    def __str__(self):
        return '%s %s %s' % (self.year, self.type, self.person)

# Utskott
class Committee(models.Model):
```

```
    year = models.IntegerField()
    type = models.ForeignKey(CommitteeType, related_name='committees')
    person = models.ForeignKey(Person, related_name='committees')

    def __str__(self):
        return '%s %s %s' % (self.year, self.type, self.person)
```

```
"""
TLKmAPI/TLKmAPI/settings.py

Django installnigar for TLKmAPI projektet.

For mera information om denna fil se
https://docs.djangoproject.com/en/1.7/topics/settings/

For en full forteckning pa installningar och varden se
https://docs.djangoproject.com/en/1.7/ref/settings/
"""

import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))


# SAKERHETS VARNING: SECRET_KEY vardet som anvands i produktion
# bor hallas hemligt
SECRET_KEY = 'dettaborvaraenlangalfanumeriskstrangmedspecialtecken'

# SAKERHETS VARNING: Anvand inte DEBUG i produktionsmiljo!
DEBUG = False

TEMPLATE_DEBUG = False

ALLOWED_HOSTS = [
    '.exempel.com',  # Tillat doman med underdomaner
]


# Applikationens definition
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'TLKdb',
    'TLKrest',
    'rest_framework',
    'corsheaders',
    'django_extensions',
)

MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
)

# Tillat Requests fran andra domaner
CORS_ORIGIN_ALLOW_ALL = True

CORS_ALLOW_METHODS = (
    'GET',
    'POST',
    'PUT',
    'PATCH',
    'DELETE',
    'OPTIONS'
)

REST_FRAMEWORK = {
    # Anvand Djangos behorighetsklasser
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.DjangoModelPermissions',
    ],
    # Autentiserings metoder
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.SessionAuthentication',
    ],
    # Filter
    'DEFAULT_FILTER_BACKENDS': [
```

```
        'rest_framework.filters.DjangoFilterBackend',
    ]
}

# URL definitionsklass
ROOT_URLCONF = 'TLKmAPI.urls'

# WSGI namn
WSGI_APPLICATION = 'TLKmAPI.wsgi.application'


# Databas
# https://docs.djangoproject.com/en/1.7/ref/settings/#databases

DATABASES = {
    # Sqlite databas
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    },

    # Exempel for Mysql
    #'mysql': {
    #    'ENGINE': 'django.db.backends.mysql',
    #    'OPTIONS': {
    #        'read_default_file': './my.cnf',
    #    },
    #}
}

# Internationalisering
# https://docs.djangoproject.com/en/1.7/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Statiska filer (CSS, JavaScript, Bilder)
```

```
# https://docs.djangoproject.com/en/1.7/howto/static-files/

STATIC_URL = '/static/'

TEMPLATE_DIRS = (
    os.path.join(BASE_DIR,  'templates'),
)
```

```
"""
TLKmAPI/TLKmAPI/urls.py
"""
from django.conf.urls import patterns, include, url
from django.contrib import admin

from TLKrest.urls import router

# URL monster
urlpatterns = patterns('',
    url(r'^', include(router.urls)),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^api-auth/',
        include('rest_framework.urls',
        namespace='rest_framework'
        )),
)
```

```
"""
TLKmAPI/TLKmAPI/wsgi.py

WSGI konfiguration for TLKmAPI projektet.

Den avslojar en WSGI modulniva variabel ``application``

For mer information om denna fil se
https://docs.djangoproject.com/en/1.7/howto/deployment/wsgi/
"""

import os, sys
```

```
# sokvag for TLKmAPI
sys.path.append('/srv/TLKmAPI/')
os.environ["DJANGO_SETTINGS_MODULE"] = "TLKmAPI.settings"

from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

```
"""
TLKmAPI/TLKrest/filters.py
"""
import django_filters
from TLKdb.models import *

# Filterklass for Person
class PersonFilter(django_filters.FilterSet):

    class Meta:
        model = Person
        fields = ['firstname', 'middlenames', 'lastname',
                  'dob', 'dod', 'email', 'joined', 'graduated',
                  'birthplace', 'city', 'zip', 'country', 'company']
```

```
"""
TLKmAPI/TLKrest/serializers.py
"""
from rest_framework import serializers

from TLKdb.models import *

# "Serializers" definierar hur API:n representeras

# Medlemstyper
class MemberTypeSerializer(serializers.ModelSerializer):

    class Meta:
        model = MemberType
        fields = ('pk', 'name')

# Styrelseposter
```

```
class BoardPositionSerializer(serializers.ModelSerializer):

    class Meta:
        model = BoardPosition
        fields = ('pk', 'name')

# Utskottstyper
class CommitteeTypeSerializer(serializers.ModelSerializer):

    class Meta:
        model = CommitteeType
        fields = ('pk', 'name')

# Funktionarstyper
class OfficialTypeSerializer(serializers.ModelSerializer):

    class Meta:
        model = OfficialType
        fields = ('pk', 'name')

# Fortjanstteckentyper
class MeritTypeSerializer(serializers.ModelSerializer):

    class Meta:
        model = MeritType
        fields = ('pk', 'name')

# Lagg till medlemmar
class MemberAddSerializer(serializers.ModelSerializer):
    type = serializers.SlugRelatedField(read_only=False, slug_field='pk',
                                        queryset=MemberType.objects.all())

    class Meta:
        model = Member
        fields = ('pk', 'person', 'year', 'type')

# Lagg till styrelsear
class BoardAddSerializer(serializers.ModelSerializer):
    type = serializers.SlugRelatedField(read_only=False, slug_field='pk',
                                        queryset=BoardPosition.objects.all())
    class Meta:
        model = Board
        fields = ('pk', 'person', 'year', 'type')

# Lagg till funktionarer
```

```
class OfficialAddSerializer(serializers.ModelSerializer):
    type = serializers.SlugRelatedField(read_only=False, slug_field='pk',
                                        queryset=OfficialType.objects.all())
    class Meta:
        model = Official
        fields = ('pk', 'person', 'year', 'type')


# Lagg till fortjansttecken
class MeritAddSerializer(serializers.ModelSerializer):
    type = serializers.SlugRelatedField(read_only=False, slug_field='pk',
                                        queryset=MeritType.objects.all())
    class Meta:
        model = Merit
        fields = ('pk', 'person', 'year', 'type')

# Lagg till utskott
class CommitteeAddSerializer(serializers.ModelSerializer):
    type = serializers.SlugRelatedField(read_only=False, slug_field='pk',
                                        queryset=CommitteeType.objects.all())
    class Meta:
        model = Committee
        fields = ('pk', 'person', 'year', 'type')

# Medlemmar
class MemberSerializer(serializers.ModelSerializer):
    type = serializers.SlugRelatedField(read_only=False, slug_field='name',
                                        queryset=MemberType.objects.all())
    class Meta:
        model = Member
        fields = ('pk', 'person', 'year', 'type')

# Styrelseposter
class BoardSerializer(serializers.ModelSerializer):
    type = serializers.SlugRelatedField(read_only=False, slug_field='name',
                                        queryset=BoardPosition.objects.all())
    class Meta:
        model = Board
        fields = ('pk', 'person', 'year', 'type')

# Funktionarer
class OfficialSerializer(serializers.ModelSerializer):
    type = serializers.SlugRelatedField(read_only=False, slug_field='name',
                                        queryset=OfficialType.objects.all())
    class Meta:
```

```
        model = Official
        fields = ('pk', 'person', 'year', 'type')

# Fortjansttecken
class MeritSerializer(serializers.ModelSerializer):
    type = serializers.SlugRelatedField(read_only=False, slug_field='name',
                                        queryset=MeritType.objects.all())
    class Meta:
        model = Merit
        fields = ('pk', 'person', 'year', 'type')

# Utskott
class CommitteeSerializer(serializers.ModelSerializer):
    type = serializers.SlugRelatedField(read_only=False, slug_field='name',
                                        queryset=CommitteeType.objects.all())
    class Meta:
        model = Committee
        fields = ('pk', 'person', 'year', 'type')

# Personer med undertabeller
class PersonSerializer(serializers.ModelSerializer):
    members = MemberSerializer(many=True, required=False, read_only=False)
    boards = BoardSerializer(many=True, required=False, read_only=False)
    officials = OfficialSerializer(many=True, required=False, read_only=False)
    merits = MeritSerializer(many=True, required=False, read_only=False)
    committees = CommitteeSerializer(many=True,
                                     required=False,
                                     read_only=False)

    class Meta:
        model = Person
        fields = ('pk', 'firstname', 'middlenames', 'lastname',
                  'dob', 'dod', 'birthplace', 'phone', 'email',
                  'address', 'city', 'zip', 'country',
                  'joined', 'graduated', 'company',
                  'company_email', 'company_phone',
                  'notes', 'members', 'boards',
                  'officials', 'merits', 'committees')

# Lagg till personer
class PersonAddSerializer(serializers.ModelSerializer):

    class Meta:
        model = Person
        fields = ('pk', 'firstname', 'middlenames', 'lastname',
```

```
                  'dob', 'dod', 'birthplace', 'phone', 'email',
                  'address', 'city', 'zip', 'country',
                  'joined', 'graduated', 'company',
                  'company_email', 'company_phone',
                  'notes',)
```

```
"""
TLKmAPI/TLKrest/urls.py
"""
from rest_framework import routers

from TLKrest.views import *

# Routrar gor det enkelt att automatiskt definiera URLar.
router = routers.DefaultRouter()

# Lagga till
router.register(r'persons/add', PersonAddViewSet, base_name='persons')
router.register(r'members/add', MemberAddViewSet, base_name='members')
router.register(r'boards/add', BoardAddViewSet, base_name='boards')
router.register(r'committees/add',
                CommitteeAddViewSet,
                base_name='committees')
router.register(r'officials/add', OfficialAddViewSet, base_name='officials')
router.register(r'merits/add', MeritAddViewSet, base_name='merits')

# Typer
router.register(r'membertypes', MemberTypeViewSet)
router.register(r'boardtypes', BoardTypeViewSet)
router.register(r'committeetypes', CommitteeTypeViewSet)
router.register(r'officialtypes', OfficialTypeViewSet)
router.register(r'merittypes', MeritTypeViewSet)

# Listningar
router.register(r'persons', PersonViewSet)
router.register(r'members', MemberViewSet)
router.register(r'boards', BoardViewSet)
router.register(r'officials', OfficialViewSet)
router.register(r'committees', CommitteeViewSet)
router.register(r'merits', MeritViewSet)
```

```
"""
TLKmAPI/TLKrest/views.py
"""
from rest_framework import viewsets, filters

from TLKrest.filters import PersonFilter
from TLKrest.serializers import *
from TLKdb.models import *


# Vy uppsattningar definierar vyernas beteende.

class PersonViewSet(viewsets.ModelViewSet):
    """
    Forteckning pa alla personer i databasen

    Anvandbara metoder: GET

    Sokning med ?search=sokord

    Giltiga sokfalt: firstname, lastname,
     birthplace, city, zip, country, company

    Filtrering med ?faltnamn=varde&frammande__faltnamn=varde
    """
    queryset = Person.objects.all()
    serializer_class = PersonSerializer
    filter_backends = (filters.SearchFilter, filters.DjangoFilterBackend)
    filter_class = PersonFilter
    search_fields = ('firstname', 'lastname',
                     'birthplace', 'city', 'zip',
                     'country', 'company')

class MemberViewSet(viewsets.ReadOnlyModelViewSet):
    """
    Forteckning pa alla medlemmar enligt primarnyckel

    Anvandbara metoder: GET

    Sokning med ?search=sokord

    Giltiga sokfalt: year, typename, lastname
```

```
    Filtrering med ?faltnamn=varde&frammande__faltnamn=varde
    """
    queryset = Member.objects.all()
    serializer_class = MemberSerializer
    filter_backends = (filters.SearchFilter,)
    search_fields = ('year', 'type__name', 'person__lastname')

class BoardViewSet(viewsets.ReadOnlyModelViewSet):
    """
    Forteckning pa alla styrelsemedlemmar enligt primarnyckel

    Anvandbara metoder: GET

    Sokning med ?search=sokord

    Giltiga sokfalt: year, typename, lastname

    Filtrering med ?faltnamn=varde&frammande__faltnamn=varde
    """
    queryset = Board.objects.all()
    serializer_class = BoardSerializer
    filter_backends = (filters.SearchFilter,)
    search_fields = ('year', 'type__name', 'person__lastname')

class CommitteeViewSet(viewsets.ReadOnlyModelViewSet):
    """
    Forteckning pa alla utskott enligt primarnyckel

    Anvandbara metoder: GET

    Sokning med ?search=sokord

    Giltiga sokfalt: year, typename, lastname

    Filtrering med ?faltnamn=varde&frammande__faltnamn=varde
    """
    queryset = Committee.objects.all()
    serializer_class = CommitteeSerializer
    filter_backends = (filters.SearchFilter,)
    search_fields = ('year', 'type__name', 'person__lastname')

class OfficialViewSet(viewsets.ReadOnlyModelViewSet):
    """
    Forteckning pa alla funktionarer enligt primarnyckel
```

```
    Anvandbara metoder: GET

    Sokning med ?search=sokord

    Giltiga sokfalt: year, typename, lastname

    Filtrering med ?faltnamn=varde&frammande__faltnamn=varde
    """
    queryset = Official.objects.all()
    serializer_class = OfficialSerializer
    filter_backends = (filters.SearchFilter,)
    search_fields = ('year', 'type__name', 'person__lastname')

class MeritViewSet(viewsets.ReadOnlyModelViewSet):
    """
    Forteckning pa alla fortjansttecken enligt primarnyckel

    Anvandbara metoder: GET

    Sokning med ?search=sokord

    Giltiga sokfalt: year, typename, lastname

    Filtrering med ?faltnamn=varde&frammande__faltnamn=varde
    """
    queryset = Merit.objects.all()
    serializer_class = MeritSerializer
    filter_backends = (filters.SearchFilter,)
    search_fields = ('year', 'type__name', 'person__lastname')

# Vy uppsattningar for tillaggning
class MemberAddViewSet(viewsets.ModelViewSet):
    """
    Lagg till medlemmar med person primarnyckel

    Anvandbara metoder: GET, PUT, UPDATE, DELETE
    """
    queryset = Member.objects.all()
    serializer_class = MemberAddSerializer

class BoardAddViewSet(viewsets.ModelViewSet):
    """
    Lagg till styrelsemedlemmar med person primarnyckel

    Anvandbara metoder: GET, PUT, UPDATE, DELETE
```

```
    """
    queryset = Board.objects.all()
    serializer_class = BoardAddSerializer

class CommitteeAddViewSet(viewsets.ModelViewSet):
    """
    Lagg till utskott med person primarnyckel

    Anvandbara metoder: GET, PUT, UPDATE, DELETE
    """
    queryset = Committee.objects.all()
    serializer_class = CommitteeAddSerializer

class OfficialAddViewSet(viewsets.ModelViewSet):
    """
    Lagg till funktionarer med person primarnyckel

    Anvandbara metoder: GET, PUT, UPDATE, DELETE
    """
    queryset = Official.objects.all()
    serializer_class = OfficialAddSerializer

class MeritAddViewSet(viewsets.ModelViewSet):
    """
    Lagg till fortjansttecken med person primarnyckel

    Anvandbara metoder: GET, PUT, UPDATE, DELETE
    """
    queryset = Merit.objects.all()
    serializer_class = MeritAddSerializer

class PersonAddViewSet(viewsets.ModelViewSet):
    """
    Lagg till nya personer i databasen

    Anvandbara metoder: GET, PUT, UPDATE, DELETE
    """
    queryset = Person.objects.all()
    serializer_class = PersonAddSerializer
    filter_backends = (filters.SearchFilter, filters.DjangoFilterBackend)
    filter_class = PersonFilter

# Viewsets for types
class MemberTypeViewSet(viewsets.ModelViewSet):
    """
```

```
    Visa, lagg till och radera medlemstyper

    Anvandbara metoder: GET, PUT, UPDATE, DELETE
    """
    queryset = MemberType.objects.all()
    serializer_class = MemberTypeSerializer

class BoardTypeViewSet(viewsets.ModelViewSet):
    """
    Visa, lagg till och radera styrelseposter

    Anvandbara metoder: GET, PUT, UPDATE, DELETE
    """

    queryset = BoardPosition.objects.all()
    serializer_class = BoardPositionSerializer

class OfficialTypeViewSet(viewsets.ModelViewSet):
    """
    Visa, lagg till och radera funktionarsposter

    Anvandbara metoder: GET, PUT, UPDATE, DELETE
    """

    queryset = OfficialType.objects.all()
    serializer_class = OfficialTypeSerializer

class CommitteeTypeViewSet(viewsets.ModelViewSet):
    """
    Visa, lagg till och radera utskott

    Anvandbara metoder: GET, PUT, UPDATE, DELETE
    """

    queryset = CommitteeType.objects.all()
    serializer_class = CommitteeTypeSerializer

class MeritTypeViewSet(viewsets.ModelViewSet):
    """
    Visa, lagg till och radera fortjanstteckentyper

    Anvandbara metoder: GET, PUT, UPDATE, DELETE
    """

    queryset = MeritType.objects.all()
```

```
    serializer_class = MeritTypeSerializer
```